ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Designing Elastic and Reliable Content Based Cloud Storage System

B Indu Priya
PG Scholor
Department of CSE,
CRIT, Anantapur.

Dr. G Prakash Babu
Professor
Department of CSE,
CRIT, Anantapur.

Abstract:
Publish/subscribe systems implemented as a service in cloud computing infrastructure provides elasticity and simplicity in composing distributed applications. Appropriate service provisioning in distributed computing infrastructure is an exigent task. Due to the dynamic changes in the rate of the live content arrival in the large scale subscriptions presents a challenge to existing publish/subscribe systems. This paper proposes ESCC (Elastic and Scalable Content based Cloud Pub/Sub System) technique that presents a framework to design elastic and reliable Content based publish/subscribe system that user single hop lookup overlay to reduce the latency in cloud computing environment. ESCC dynamically adjust the scale of the servers depending on the churn workloads. ESCC achieves high throughput rate when compared to various workloads.

Key words: ***Publish / Subscribe, Cloud storage, Scalable, content based retrieval, subscriptions***

I. Introduction

Cloud computing is the embryonic paradigm with changing definitions but for this research work, we can define in the term of a virtual infrastructure which will provide shared information and communication technology services, via an internet “cloud,” for “multiple external users” by the use of the Internet or “large-scale private networks.”[1] A computer user access can be provided to Information Technology services i.e., applications, servers, data storage, there is no need to understand the technology or also ownership of the infrastructure. An analogy to an electricity computing grid is to be useful for comprehend cloud computing. A power company maintains and owns the infrastructure, a distribution company disseminates the electricity, and the consumers merely use the resources without the ownership or operational responsibilities. [2]. It is a subscription-based service where networked storage space and computer resources can be obtained. One way to think of cloud computing is to be considered our experience with email. As the real-time requirement of data dissemination becomes increasingly significant in many fields, the emergency applications have received increasing attention, for instance, stock quote distribution, earthquake monitoring [1], emergency weather alert [2], smart transportation system [3], and social networks. Recently, the development of emergency applications demonstrates two trends. One is the sudden change of the arrival live content rate. Take ANSS [1] as an example, its mission is to provide real-time and accurate seismic information for emergency response personnel.

Publish/subscribe (pub/sub) paradigm is a key technology for asynchronous data dissemination that are widely used in the emergency applications. It decouples senders and receivers of the emergency applications in space, time, and synchronization [5], which enables a pub/sub system to seamlessly expand to massive size. However, traditional pub/sub system faces a number of challenges. Firstly, the system must guarantee real-time event matching capacity when it expands to very large-scale. For instance, Facebook contains billions of users and 684,478 pieces of content are published on average in every minute [6]. Secondly, the system needs to be elastic to the sudden change of incoming event rate to achieve high performance–price ratio. This is because if a fixed number of servers are deployed in response to the sudden change of incoming event rate, numerous servers are in the idle states delivering few messages in most of the time. Thirdly, the service must be tolerant to the server failures. In emergency applications, a large number of machines and links may be unavailable instantaneously due to hardware errors or operator mistakes, which leads to the loss of events and subscriptions.

Millions of messages are generated by sensors in a short time when an earthquake happens, while few events are generated if there is no earthquake. The other is the skewness of the large-scale subscriptions. That is, a large number of subscribers demonstrate similar interests. For instance, the dataset [4] of 297 K users of Facebook shows that the hottest 100 topics together have more than 1.1 million subscribers, while 71% topics have no more than 16 subscribers. In contrast, a large number of P2P based systems [13] do not provide dedicated brokers. All nodes are organized into a P2P based overlay [6], and they act both as publishers and subscribers. All events and subscriptions are forwarded through multihop routing. Then the subscriptions falling into the same subspace of the entire content space are organized into a multicast group or stored in a rendezvous node. With the growth of arrival event rate, the multi-hop routing may lead to high latency and traffic overhead. A large body of skewed subscriptions may incur unbalanced load on the multicast groups or rendezvous nodes, which imposes a limit on scalability. Besides that, the P2P based systems are hard to provide elastic service due to the unpredictability of the node behavior.

Existing pub/sub systems are not adequate to efficiently address all above challenges. In the broker based pub/sub systems [7–14], all publishers and subscribers are directly connected to a group of servers, known as brokers. Subscriptions are commonly replicated to all brokers or a part of brokers, so that each broker can match events and forward them to the interested subscribers. However, replicating subscriptions incurs that each event is matched against the same subscriptions for many times, which leads to high matching latency and low scalability when a large number of events and subscriptions arrive. Moreover, it is difficult to provide elastic service in order to deal with the changing workloads. This is because these systems often over-provision brokers to reduce their loads, and do not have financial incentives to reduce the scale of brokers during off-peak hours

2. Related Work

This section presents some elasticity solutions implemented in IaaS clouds. In general, most public cloud providers offer some elasticity feature, from the most basic, to more elaborate automatic solutions. In turn, the solutions developed by academy are similar to those provided by commercial providers, but include new techniques and methodologies for elastic provisioning of resources. Amazon Web Services [14], one of the most traditional cloud providers, offers a replication mechanism called AutoScaling, as part of the EC2 service. The solution is based on the concept of Auto Scaling Group (ASG), which consists of a set of instances that can be used for an application. Amazon Auto-Scaling uses an automatic-reactive approach, in which, for each ASG there is a set of rules that defines the instances number to be added or released. The metric values are provided by Cloud Watch monitoring service, and include CPU usage, network traffic, disk reads and writes. The solution also includes load balancers that are used to distribute the workload among the active instances.

GoGrid [10] and Rackspace [11] also implement replication mechanisms, but unlike Amazon, does not have native automatic elasticity services. Both providers offer API to control the amount of virtual machines instantiated, leaving to the user the implementation of more elaborate automated mechanisms. To overcome the lack of automated mechanisms, tools such as RightScale [6] and Scalr [7] have been developed. RightScale is a management platform that provides control and elasticity capabilities for different public cloud providers (Amazon, Rackspace, GoGrid, and others) and also for private cloud solutions (CloudStack, Eucalyptus and OpenStack). The solution provides automatic-reactive mechanisms based on an Elasticity Daemon whose function is to monitor the input queues, and to launch worker instances to process jobs in the queue. Different scaling metrics (from hardware and applications) can be used to determine the number of worker instances to launch and when to launch these instances. Scalr is an open-source project whose goal is to offer elasticity solutions for web applications that supports several clouds, such as, Amazon, Rackspace, Eucalyptus and Cloudstack. Currently supports Apache and ngnix, MySQL database, PostgreSQL, Redis and MongoDB. Likewise RightScale, the operations use hardware and software monitoring metrics to trigger actions. A more comprehensive elasticity solution is provided by OnApp Cloud [5], a software package for IaaS cloud providers. According to its documentation, it is possible implement replication and redimensioning of VM's, allowing changes in virtual environments manually or automatically, using user-defined rules and metrics obtained by the monitoring system. In order to take full advantage of the elasticity provided by clouds, it is necessary more than an elastic infrastructure. It is also necessary that the applications have the ability to dynamically adapt itself according to changes in its requirements. In general, applications developed in PaaS clouds have implicit elasticity. These clouds provide execution environments, called containers, in which users can execute their applications without having to worry about which resources will be used. In this case, the cloud manages automatically the resource allocation, so developers do not have to constantly monitor the service status or interact to request more resources [8], [9]. An example of PaaS platform with elasticity support is Aneka [10]. In Aneka, when an application needs more resources, new container instances are executed to handle the demand, using local or public cloud resources. There are exceptions, such as Microsoft Azure [12] in which the user must define the resources used by applications. Some academic works have presented elasticity mechanisms for applications, with the main objective of enabling the development of flexible and adaptable applications for cloud environments.

Neamtiu [13] described Elastin, a framework that comprises a compiler and a runtime environment, whose goal is to convert inelastic programs into elastic applications. The idea behind Elastin is the use of a compiler that combines diverse program versions into a single application that can switch between configurations at runtime, without shutting down the application. The executable binary file stores several configurations, each one for a given scenario. The choice of which configuration should be used is defined by the user and can be changed at runtime. Vijayakumar et al. [9] presented an elasticity mechanism for streaming applications. The proposal consists in to adapt the CPU resources of the virtual machine in accordance with the data streams. The streaming application consists of a pipeline of several stages, each one allocated individually in a virtual machine. The elasticity mechanism compares the input and output flow at each stage, and if there is a bottleneck, increases the percentage of physical CPU allocated to the virtual machine that hosts the affected stage. Knauth & Fetzer [4] also addressed streaming applications, but focusing energy consumption reduction. The proposed solution employs virtual machine migration and consolidation to provide elasticity. The basic idea is to start each application stage inside a virtual machine. When load is minimal, all virtual machines are consolidated into a minimal set of physical machines. When the load increases, virtual machines are migrated to other servers, until each physical server hosts a single virtual machine. Rajan et al. [14] presented Work Queue, a framework for the development of master-slave elastic applications. Applications developed using Work Queue allow

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

adding slave replicas at runtime. The slaves are implemented as executable files that can be instantiated by the user on different machines on demand. When executed, the slaves communicate with the master that coordinates task execution and the data exchange.

3. PROPOSED SCHEME:

3.1 ESCC Technique: Elastic and Scalable Content based Cloud Pub/Sub System

To achieve scalable and elastic total order in content based pub/sub system, we first propose a distributed two-layer pub/sub framework based on the cloud computing environment. At a high level, the framework consists of two layers: the matching layer and the delivery layer

- The matching layer is responsible for matching events against subscriptions and sending events with matched subscribers to the delivery layer.
- The delivery layer is responsible for ordering events according to the total order semantics, and distributing them to their interested subscribers For matching service, the main factors limiting its scalability contain the scale of subscriptions, the distributions of subscriptions and events, data dimensionality, and the arrival rate of events. For total ordering service, the main factors limiting its scalability contain the arrival rate of events and the probability of ordering conflict. Thus, we can flexibly improve the capacities of either event matching service or total ordering service based on various workload characteristics if both services are decoupled.

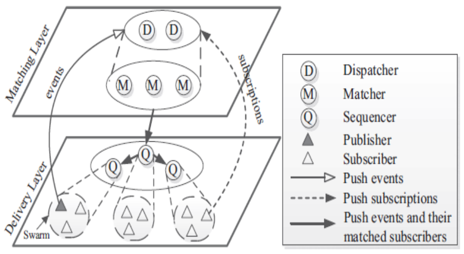

Figure 1.ESCC Architecture

3.2 Matching Layer

The matching layer is responsible for matching events against subscription and sending events with their matched subscribers to the delivery layer. At this layer, we employs SREM technique as shown in Figure 1 to implement high matching throughput. In SREM, through a hierarchical multi-attribute space partition technique (called HPartition), the content space is divided into multiple hypercube, each of which is managed by one server. Subscriptions and events falling into the same hypercube are matched with each other, such that the matching latency can be greatly reduced. Besides, a performance aware detection technique (called PDetection) is proposed in ESCC to adaptively adjust the scale of servers based on the churn workloads

3.2 Delivery Layer

The delivery layer is responsible for total ordering events and delivering them to their interested subscribers. The main novelty of this layer lies in a preceding graph building technique (called PGBuilder) and a performance aware provisioning technique (called PProvision). The first aims to reduce the total order latency in a scalable manner. In PGBuilder, subscribers are divided into multiple groups, each of which is managed by a single server. That is, all servers of PGBuilder are able to detect total order conflicts among events simultaneously, which greatly reduces the delivery latency. Each server of PGBuilder constructs preceding graphs among arrival events, which can quickly detect non-conflicting events and deliver them in a parallel manner. Besides, to ensure reliable delivery, PGBuilder provides a series of dynamics maintenance mechanisms.

Algorithm 1 : PGBuilder

```
Input: New Arrival event 'e'
Q: The global queue of sequencer
C: Cluster ,CPL : Cluster preceding list
e: Direct, DPL: Direct Preceding List

    1. Initialize cluster C with Q
    2. if C==0 then initialize C as new
       Cluster
    3. Process the list L as CPL and if size
       (CPL) != 0 the
```

```
4.  add DPL.get(Q).
5.  int i =0
6.  while (i < tmpList.Size()) do
7.  e = tmpList.get(i)
8.  for(int k =0; k< e.DPL.size(); K++)
    do
9.  tempE = e.DPL.get(K);
10. if( ! tmpList.contains(tempE) ) then
11. tmpList.add(tempE)
12. i++
```

Firstly, the arrival events in each sequencer are dispatched into multiple separated clusters, and all these clusters are managed by a global queue (GQ). Then, e is added into the tail cluster. Clusters in the GQ are naturally conflicted with each other, and each cluster can be treated as a sliding window. That is, each sequencer only processes the head cluster. After the events of the head cluster are delivered to all their corresponding subscribers, the head cluster is removed from the global queue, and the sequencer processes next head cluster. Thus, each new arrival event only needs to detect conflicts against the events in the tail cluster of GQ, but not all events of GQ. Through dividing events into multiple clusters, it greatly reduces the conflict detecting latency regardless of the event arrival rate.

3.3 Delivery Strategy

In ESCC technique, there are mainly two roles: sequencers and subscribers. Since the joining or leaving of both roles may severely hurt the performance of total ordering, we will discuss how to keep continuous and efficient total ordering under dynamic networks.

i. Subscriber:

Recall that each subscriber sends its subscriptions to one of the dispatchers in our framework as shown in Fig. 1. When a new subscriber joins the system, it is dispatched to one sequencer whose hash value is nearest to its own value according to the consistent hashing technique. To ensure reliable delivery, the sequencer starts to send the next event until it receives all acknowledgements from subscribers of the last event. Thus, sequencers need to obtain the latest view of its local subscribers. Otherwise, waiting acknowledgements from failed subscribers may lead to high delivery latency.

Algorithm 2: Event Delivery

```
Input:  e: Delivering event
/* C.CPL: the cluster preceding list of cluster C*/
/* e.DPL: the direct predecessor list of e */
1. **foreach** (subscriber s in Dest(e)) **do**
2. send(s,e);
3. C.CPL. remove(e);
4. if C.CPL.isEmpty() then
5. ConflictResolution()
6. else
7. foreach(event e in e,DSL) do
8. e.DPL.remove(e);
9. Delivery(e)
```

ii. Sequencer:

When a number of new sequencers join in the system, the root sequencer assigns every new sequencer to be a child of the existing sequencers one by one. Correspondingly, each exiting sequencer should detect whether its every local subscriber needs to be migrated to one of the new sequencers based on the consistent hashing technique.

4. Performance Analysis

This section presents the design and implementation of ESCC prototype and performance analysis of the proposed framework. In order to design the prototype in modular and portable fashion we made use of the object oriented middleware that allows the users to focus on the application logic, rather than interact with lowlevel network programming interfaces.

To evaluate the scalability and elasticity each matcher estimates its waiting time per 200 ms. we set both timeout intervals Tout and T′ out to 10 s for adding and removing matchers, respectively That is, ESCC collects all new matchers or failed matchers until the continuous arrival interval or failure interval of two matchers exceeds 10 s. Besides that, Ds supposes that a matcher fails if its continuous arrival interval of two heartbeat messages exceeds 10 s Firstly the evaluation process involves in how how ESCC adjust the number of matchers to adapt to both linear and instantaneous increasing event arrival rates as shown in Figure 2. Next the process show the elasticity of ESCC adapts to both linear and instantaneous decreasing event arrival rates.

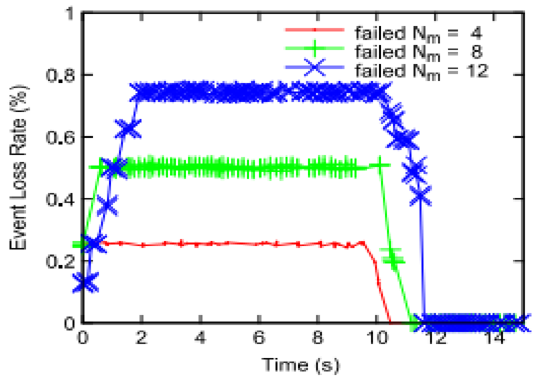

Figure 2. The changing of event loss rate with a number of matcher's failure.

5. Conclusion

This paper introduces ESCC, a novel scalable and elastic event matching approach for the attribute-based pub/sub systems. ESCC utilizes an one-hop lookup overlay in the cloud computing environment to reduce the clustering latency. Through a hierarchical multi-attribute space partition technique, ESCC reaches scalable clustering of subscriptions and matches each event on one cluster. The performance-aware detection technique enables the system to adaptively adjusts the scale of matchers according to the changing of workloads. Compared with the existing cloud based pub/sub systems, our analytical and experimental results demonstrate that ESCC shows a much higher matching rate and better load balance with different workload characteristics. Moreover, ESCC adapts to the sudden workload changes and server failures with low latency and small traffic overhead.

References:

1. M. Gjoka, M. Kurant, C.T. Butts, A. Markopoulou, Walking in Facebook: a case study of unbiased sampling of OSNs, in: International Conference on Computer Communications, INFOCOM, 2010, pp. 1–9.

2. P.T. Eugster, P. Felber, R. Guerraoui, A.-M. Kermarrec, The many faces of publish/subscribe, ACM Computing Surveys (CSUR) 35 (2) (2003) 114–131.

3. Datacreatedperminite. URL: http://www.domo.com/blog/2012/06/ how-much-data-is-created-every-minute/?dkw=socf3/.

4. P. Pietzuch, J. Bacon, Hermes: a distributed event-based middleware architecture, in: 22nd International Conference on Distributed Computing Systems Workshops, 2002.

5. F. Cao, J.P. Singh, Efficient event routing in content-based publish/subscribe service network, in: International Conference on Computer Communications, INFOCOM, 2004.

6. G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, D.C. Sturman, An efficient multicast protocol for content-based publish–subscribe systems, in: IEEE International Conference on Distributed Computing Systems, ICDCS, 1999, pp. 262–272.

7. F. Cao, J.P. Singh, Medym: match-early with dynamic multicast for contentbased publish–subscribe networks, 2005, pp. 292–313.

8. L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, D. Sturman, Exploiting IP multicast in content-

based publish–subscribe systems, in: IFIP/ACM International Conference on Distributed Systems Platforms, 2000, pp. 185–207.

9. A. Riabov, Z. Liu, J.L. Wolf, P.S. Yu, L. Zhang, Clustering algorithms for content-based publication–subscription systems, in: IEEE 22nd International Conference on Distributed Computing Systems, ICDCS, 2002, pp. 133–142.

10. A. Carzaniga, Architectures for an event notification service scalable to widearea networks, Ph.D. Thesis, POLITECNICO DI MILANO, 1998.

11.Y.-M. Wang, L. Qiu, C. Verbowski, D. Achlioptas, G. Das, P.-Å Larson, Summarybased routing for content-based event distribution networks, Computer Communication Review 34 (5) (2004) 59–74.

12.A. Carzaniga, M.J. Rutherford, A.L. Wolf, A routing scheme for content-based networking, in: IEEE International Conference on Computer Communications, INFOCOM, 2004.

13.W.W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, A.P. Buchmann, A peer-topeer approach to content-based publish/subscribe, in: Proceedings of the 2nd International Workshop on Distributed Event-Based Systems, 2003, pp. 1–8.

14.I. Aekaterinidis, P. Triantafillou, Pastrystrings: a comprehensive content-based publish/subscribe DHT network, in: IEEE 26nd International Conference on Distributed Computing Systems, ICDCS, 2006